Frankenstein's PSO: A Composite Particle Swarm Optimization Algorithm

Marco A. Montes de Oca, Thomas Stützle, Mauro Birattari, *Member, IEEE*, and Marco Dorigo, *Fellow, IEEE*

***Abstract*— During the last decade, many variants of the original particle swarm optimization (PSO) algorithm have been proposed. In many cases, the difference between two variants can be seen as an algorithmic component being present in one variant but not in the other. In the first part of the paper, we present the results and insights obtained from a detailed empirical study of several PSO variants from a component difference point of view. In the second part of the paper, we propose a new PSO algorithm that combines a number of algorithmic components that showed distinct advantages in the experimental study concerning optimization speed and reliability. We call this composite algorithm Frankenstein's PSO in an analogy to the popular character of Mary Shelley's novel. Frankenstein's PSO performance evaluation shows that by integrating components in novel ways effective optimizers can be designed.**

***Index Terms*— Continuous optimization, experimental analysis, integration of algorithmic components, particle swarm optimization (PSO), run-time distributions, swarm intelligence.**

I. Introduction

Since particle swarm optimization (PSO) was introduced [1], many modifications to the original algorithm have been proposed (for reviews see [2]–[4]). In many cases, the modifications can be seen as algorithmic components that provide an improved performance. These algorithmic components range from added constants in the particles' velocity-update rule [5] to stand-alone algorithms that are used as components of hybrid PSO algorithms [6].

In this paper, we first present the results of an experimental study of various PSO algorithms. Our comparison focuses on the differences between mechanisms for updating a particle's velocity, although other factors such as the selection of the population topology, the number of particles, and the strategies for updating at run time various parameters that influence performance are also considered. The comparison of PSO variants is performed with their most commonly used parameter settings (i.e., those commonly found in the literature). The experimental setup and the choice of the PSO variants allow the identification of performance differences that can be ascribed to specific algorithmic components and their interactions and, hence, contribute to an improved understanding of the PSO approach.

In the second part of the paper, we design and evaluate a new composite PSO algorithm called Frankenstein's PSO, which combines algorithmic components that we have identified as contributing positively to either convergence speed or optimization reliability of PSO algorithms. The final comparison of Frankenstein's PSO with the PSO variants studied in the first part of the paper shows that, by integrating already existing components in novel ways, effective optimizers can be designed.

From a wider perspective, this paper adds evidence that a careful experimental study of algorithm components and their interactions can be a crucial step toward a more directed design of new high-performing composite (or hybrid) algorithms.

II. Particle Swarm Optimization Algorithms

To optimize a d-dimensional continuous objective function $f : \mathbb{R}^d \to \mathbb{R}$, a population of particles $\mathcal{P} = \{p_1, \ldots, p_n\}$ (called *swarm*) is randomly initialized in the solution space. The objective function determines the quality of the solution represented by a particle's position. (Without loss of generality, we restrict the following discussion to minimization problems.)

At any time step t, a particle p_i has an associated position vector $\boldsymbol{x}_i^t$ and a velocity vector $\boldsymbol{v}_i^t$. A vector $\boldsymbol{pb}_i^t$ (known as *personal best*) stores the best position the particle has ever visited. Particle p_i is said to have a topological neighborhood $\mathcal{N}_i \subseteq \mathcal{P}$ of particles. The best *personal best* vector in a particle's neighborhood (called *local best*) is a vector $\boldsymbol{lb}_i^t$ such that $f(\boldsymbol{lb}_i^t) \leq f(\boldsymbol{pb}_j^t)\, \forall p_j \in \mathcal{N}_i$.

PSO algorithms update the particles' velocities and positions iteratively until a stopping criterion is met. The basic velocity- and position-update rules are

$$\boldsymbol{v}_i^{t+1} = \boldsymbol{v}_i^t + \varphi_1 \boldsymbol{U}_1^t(\boldsymbol{pb}_i^t - \boldsymbol{x}_i^t) + \varphi_2 \boldsymbol{U}_2^t(\boldsymbol{lb}_i^t - \boldsymbol{x}_i^t) \tag{1}$$

and

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1} \tag{2}$$

where φ_1 and φ_2 are two parameters called acceleration coefficients, $\boldsymbol{U}_1^t$ and $\boldsymbol{U}_2^t$ are two $d \times d$ diagonal matrices with in-diagonal elements distributed in the interval $[0, 1)$ uniformly at random. (These matrices are generated at every iteration.)

Manuscript received March 21, 2007; revised November 5, 2007, July 29, 2008, January 22, 2009, and April 2, 2009; accepted April 9, 2009. Current version published September 30, 2009. The work described in this paper was supported in part by the ANTS and by the META-X projects, two *Actions de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. The work of M. A. Montes de Oca was supported by the Programme Alßan, the European Union Programme of High Level Scholarships for Latin America, under Scholarship E05D054889MX. The work of T. Stützle, M. Birattari, and M. Dorigo was supported by the fund for scientific research, F.R.S-FNRS, of the French Community of Belgium.

The authors are with the Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle, Université Libre de Bruxelles, 1050 Brussels, Belgium (e-mail: mmontes@ulb.ac.be; stuetzle@ulb.ac.be; mbiro@ulb.ac.be; mdorigo@ulb.ac.be).

Digital Object Identifier 10.1109/TEVC.2009.2021465

A maximum velocity parameter $V_{\max}$ prevents velocities from growing to extremely large values [7], [8].

In the following paragraphs, we describe the variants that were selected to be part of our study. For practical reasons, many variants had to be left out; however, the selection allows the study of a number of different PSO algorithmic components including those that, for us, are among the most influential or promising ones.

A. Constricted Particle Swarm Optimizer

Clerc and Kennedy [5] added a *constriction factor* to the particles' velocity-update rule to avoid the unlimited growth of the particles' velocity. Equation (1) is modified to

$$\boldsymbol{v}_i^{t+1} = \chi \left(\boldsymbol{v}_i^t + \varphi_1 \boldsymbol{U}_1^t(\boldsymbol{pb}_i^t - \boldsymbol{x}_i^t) + \varphi_2 \boldsymbol{U}_2^t(\boldsymbol{lb}_i^t - \boldsymbol{x}_i^t)\right) \quad (3)$$

with $\chi = 2/|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|$ where χ is the constriction factor, $\varphi = \sum_i \varphi_i$, and $\varphi > 4$. Usually, φ_1 and φ_2 are set to 2.05, giving as a result χ equal to 0.729 [8], [9]. This variant will be referred to as *constricted* PSO in the rest of the paper.

B. Time-Varying Inertia Weight Particle Swarm Optimizers

Shi and Eberhart [10], [11] noticed that the first term of the right-hand side of (1) plays the role of a particle's "inertia" and they introduced the idea of an *inertia weight*. The velocity-update rule was modified to

$$\boldsymbol{v}_i^{t+1} = w^t \boldsymbol{v}_i^t + \varphi_1 \boldsymbol{U}_1^t(\boldsymbol{pb}_i^t - \boldsymbol{x}_i^t) + \varphi_2 \boldsymbol{U}_2^t(\boldsymbol{lb}_i^t - \boldsymbol{x}_i^t) \quad (4)$$

where w^t is the time-dependent inertia weight. Shi and Eberhart proposed to set the inertia weight according to a time-decreasing function so as to have an algorithm that initially explores the search space and only later focuses on the most promising regions. Experimental results showed that this approach is effective [7], [10], [11]. The function used to schedule the inertia weight is defined as

$$w^t = \frac{wt_{\max} - t}{wt_{\max}}(w_{\max} - w_{\min}) + w_{\min} \quad (5)$$

where $wt_{\max}$ marks the time at which $w^t = w_{\min}$; $w_{\max}$ and $w_{\min}$ are the maximum and minimum values the inertia weight can take, respectively. Normally, $wt_{\max}$ coincides with the maximum time allocated for the optimization process. We identify this variant as *decreasing-IW* PSO. The constricted PSO is a special case of this variant but with a constant inertia weight. We treat them as different variants because of their different behavior and for historical reasons.

Zheng *et al.* [12], [13] experimented with a time-increasing inertia weight function, obtaining, in some cases, better results than the decreasing-IW variant. Concerning the schedule of the inertia weight, Zheng *et al.* also used (5), except that the values of $w_{\max}$ and $w_{\min}$ were interchanged. This variant is referred to as *increasing-IW* PSO.

Eberhart and Shi [14] proposed a variant in which an inertia weight vector is randomly generated according to a uniform distribution in the range [0.5, 1.0) with a different inertia weight for each dimension. This range was inspired by Clerc and Kennedy's constriction factor because the expected value of the inertia weight in this case is $0.75 \approx 0.729$. Accordingly, in this *stochastic-IW* PSO algorithm, acceleration coefficients are set to the product of $\chi \cdot \varphi_i$ with $i \in \{1, 2\}$.

C. Fully Informed Particle Swarm Optimizer

Mendes *et al.* [15] proposed the fully informed particle swarm (FIPS), in which a particle uses information from all its topological neighbors. Clerc and Kennedy's constriction factor is also adopted in FIPS; however, the value φ (i.e., the sum of the acceleration coefficients) is equally distributed among all the neighbors of a particle.

For a given particle p_i, φ is decomposed as $\varphi_k = \varphi/|\mathcal{N}_i|$, $\forall p_k \in \mathcal{N}_i$. The velocity-update equation becomes

$$\boldsymbol{v}_i^{t+1} = \chi \left[\boldsymbol{v}_i^t + \sum_{p_k \in \mathcal{N}_i} \varphi_k \boldsymbol{U}_k^t(\boldsymbol{pb}_k^t - \boldsymbol{x}_i^t)\right]. \quad (6)$$

D. Self-Organizing Hierarchical Particle Swarm Optimizer With Time-varying Acceleration Coefficients

Ratnaweera *et al.* [16] proposed the self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients (HPSOTVAC), in which the inertia term in the velocity-update rule is eliminated. Additionally, if any component of a particle's velocity vector becomes zero (or very close to zero), it is reinitialized to a value proportional to $V_{\max}$, which is the maximum velocity allowed. This gives the algorithm a local search behavior that is amplified by linearly adapting the value of the acceleration coefficients φ_1 and φ_2. The coefficient φ_1 is decreased from 2.5 to 0.5, and the coefficient φ_2 is increased from 0.5 to 2.5. In HPSOTVAC, the maximum velocity is linearly decreased during a run so as to reach 1/10 of its value at the end. A low reinitialization velocity near the end of the run allows particles to move slowly near the best region they have found. The resulting PSO variant is a kind of local search algorithm with occasional magnitude-decreasing unidimensional restarts.

E. Adaptive Hierarchical Particle Swarm Optimizer

The adaptive hierarchical PSO (AHPSO) [17] modifies the neighborhood topology at run time. It uses a tree-like topology structure in which particles with better objective function evaluations are located in the upper nodes of the tree. At each iteration, a child particle updates its velocity considering its own previous best performance and the previous best performance of its parent. Before the velocity-update process takes place, the previous best fitness value of any particle is compared with that of its parent. If it is better, child and parent swap their positions in the hierarchy. Additionally, AHPSO adapts the branching degree of the tree while solving a problem to balance the exploration-exploitation behavior of the algorithm: a hierarchy with a low branching degree has a more exploratory behavior than a hierarchy with a high branching degree. In AHPSO, the branching degree is decreased by k_{adapt} degrees (one at a time) until a certain minimum degree $d_{\min}$ is reached. This process takes place every f_{adapt} number of iterations. For more details, see [17].

III. Experimental Setup

The complete experimental design examines five factors.

1) **PSO algorithm**. This factor considers the differences between PSO variants. Specifically, we focused on 1) different strategies for updating inertia weights, 2) the use of static and time-varying population topologies, and 3) different strategies for updating a particle's velocity.
2) **Problem**. We selected some of the most commonly used benchmark functions in experimental evolutionary computation. Since most of these functions have their global optimum located at the origin, we shifted it to avoid any possible search bias as suggested by Liang *et al.* [18]. In most cases, we used the shift values proposed in the set of benchmark functions used for the special session on real parameter optimization of the IEEE CEC 2005 [19]. Table I lists the benchmark functions used in our study. In all cases, we used their 30-dimensional versions. [Their definitions can be found in this paper's supplementary information web page [20].[1]] All algorithms were run 100 times on each problem.
3) **Population topology**. We use three of the most commonly used population topologies: The fully connected topology, in which every particle is a neighbor of any other particle in the swarm; the von Neumann topology, in which each particle is a neighbor of four other particles; and the ring topology, in which each particle is a neighbor of another two particles. In our setup, all particles are also neighbors to themselves. These three topologies are tested with all variants except in the case of AHPSO which uses a time-varying topology. The selected topologies provide different degrees of connectivity between particles. The goal is to favor exploration in different degrees: The less connected is a topology, the more it delays the propagation of the best-so-far solution. Thus, low connected topologies result in more exploratory behavior than highly connected ones [21]. Although recent research suggests that random topologies can be competitive to predefined ones [22], they are not included in our setup in order not to have an unmanageable number of free variables.
4) **Population size**. We considered three population sizes: 20, 40, and 60 particles. With low connected topologies and large populations, the propagation of information is slower and thus it is expected that a more "parallel" search takes place. The configurations of the von Neumann topologies for 20, 40, and 60 particles were, respectively, 5×4, 5×8, and 6×10 particles. The population is initialized uniformly at random over the ranges specified in Table I. Since the problems' optima were shifted, the initialization range is asymmetric with respect to them.
5) **Maximum number of function evaluations**. This factor determined the stopping criterion. The limit was set to 10^6 function evaluations. However, data were collected during a run to determine relative performances for shorter runs. The goal was to find variants that are well suited for different application scenarios. The first two cases (10^3 and 10^4 function evaluations) model scenarios in which there are scarce resources and the best possible solution is sought given a restrictive time limit. The other two cases (10^5 and 10^6 function evaluations) model scenarios in which the main concern is to find high quality solutions without paying too much attention to the time it takes to find them.

[1] At this same address the reader can find all the supporting supplementary information (definitions, tables, and graphs) that, for the sake of conciseness, we do not present here.

TABLE I
BENCHMARK PROBLEMS

Function name	Search range	Modality
Ackley	$[-32.0, 32.0]^n$	Multimodal
Griewank	$[-600.0, 600.0]^n$	Multimodal
Rastrigin	$[-5.12, 5.12]^n$	Multimodal
Salomon	$[-100.0, 100.0]^n$	Multimodal
Schwefel (sine root)	$[-512.0, 512.0]^n$	Multimodal
Step	$[-5.12, 5.12]^n$	Multimodal
Rosenbrock	$[-30.0, 30.0]^n$	Unimodal
Sphere	$[-100.0, 100.0]^n$	Unimodal

In our experimental setup, each algorithm was run with the same parameter settings across all benchmark problems. When possible, we use the most commonly used parameter settings found in the literature. These parameter settings are listed in Table II.

In our experimental analysis, we examined the algorithms' performance at different levels of aggregation. At a detailed level, we analyze the algorithms' qualified run-length distributions (RLDs, for short). At a more aggregate level, we use the median solution quality reached by the algorithms at different stopping criteria. The most important elements of the RLD methodology are explained below (for a detailed exposition, see [23]).

The number of function evaluations a stochastic optimization algorithm needs to find a solution of a certain quality on a given problem can be modeled as a random variable. Its associated cumulative probability distribution $RL_q(l)$ is the algorithm's RLD, defined as

$$RL_q(l) = P(L_q \leq l) \tag{7}$$

where L_q is the random variable representing the number of function evaluations needed to find a solution of quality q, and $P(L_q \leq l)$ is the probability that L_q takes a value less than or equal to l function evaluations. Theoretical RLDs can be estimated empirically using multiple independent runs of an algorithm.

An empirical RLD provides a graphical view of the development of the probability of finding a solution of a certain quality as a function of time. When this probability does not increase, or it does but very slowly, the algorithm is said to stagnate. In this paper we use the word stagnation to refer to the phenomenon of slow or no increment of the probability of finding a solution of a specific quality. Note that no reference

TABLE II
PARAMETER SETTINGS

Algorithm	Settings
Constricted	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.05$. Constriction factor $\chi = 0.729$. Maximum velocity $V_{\max} = \pm X_{\max}$, where $X_{\max}$ is the maximum of the search range.
Decreasing-IW	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.0$. Linearly-decreasing inertia weight from 0.9 to 0.4. The final value is reached at the end of the run. Maximum velocity $V_{\max} = \pm X_{\max}$.
Increasing-IW	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.0$. Linearly-increasing inertia weight from 0.4 to 0.9. The final value is reached at the end of the run. Maximum velocity $V_{\max} = \pm X_{\max}$.
Stochastic-IW	Acceleration coefficients $\varphi_1 = \varphi_2 = 1.494$. Uniformly distributed random inertia weight in the range [0.5, 1.0). Maximum velocity $V_{\max} = \pm X_{\max}$.
FIPS	Acceleration parameter $\varphi = 4.1$. Constriction factor $\chi = 0.729$. Maximum velocity $V_{\max} = \pm X_{\max}$.
HPSOTVAC	Acceleration coefficient φ_1 linearly decreased from 2.5 to 0.5 and coefficient φ_2 linearly increased from 0.5 to 2.5. Linearly decreased reinitialization velocity from $V_{\max}$ to $0.1 \cdot V_{\max}$. Maximum velocity $V_{\max} = \pm X_{\max}$.
AHPSO	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.05$. Constriction factor $\chi = 0.729$. Initial branching factor is set to 20, $d_{\min}$, f_{adapt}, and k_{adapt} were set to 2, $1000 \cdot m$, and 3, respectively, where m is the number of particles.

to the state of the optimization algorithm is implied (e.g., in active search or otherwise).

In stagnation cases, the probability of finding a solution of a certain quality may be increased by restarting the algorithm at fixed cut-off times without carrying over information from the previous runs [23]. These *independent* restarts entail re-running the algorithm using a different random seed. However, the output of the algorithm with restarts is always the overall best-so-far solution across all independent runs.

The RLD of the algorithm with periodic restarts will approximate, in the long run, an exponential distribution. However, independent restarts can be detrimental if an algorithm's original RLD grows faster than an exponential distribution. Given an algorithm's RLD, it is possible to estimate the number of function evaluations needed for finding a solution of a required quality with a probability greater than or equal to z supposing an optimal restart policy. This estimation is sometimes called *computational effort* [24] and it is defined as

$$effort = \min_{l} \left\{ l \cdot \frac{\ln(1-z)}{\ln(1 - RL_q(l))} \right\}. \qquad (8)$$

We use this measure to account for the possibility of restarting the compared algorithms with optimal restart policies.

Another measure that will be used in the description of the results is the *first hitting time* H_q for a specific solution quality q. H_q is an estimation of the minimum number of evaluations that an algorithm needs for finding a solution of a quality level q. It is defined as

$$H_q = \min\{l \geq 0; RL_q(l) > 0\}. \qquad (9)$$

IV. Performance Comparison of PSO Algorithms

The comparison is carried out in three phases. In the first one, a problem-dependent run-time behavior comparison based on RLDs is performed (a preliminary series of results is published in [25]). In the second phase, data from all the problems of our benchmark suite are aggregated and analyzed. In the third phase, we study the effects of using different inertia weight schedules on the performance of the concerned variants. Results that are valid for all the tested problems are explicitly summarized.

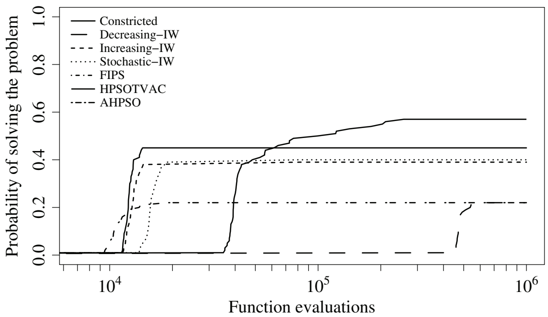
A. Results: Run-Length Distributions

The graphs presented in this section show a curve for each of the compared algorithms corresponding to a particular combination of a population topology and a population size. Since AHPSO does not use a fixed topology, its RLDs are the same across topologies and its results can therefore be used as a reference across plots for a same problem. The RLDs we present here were obtained using swarms of 20 and 60 particles.

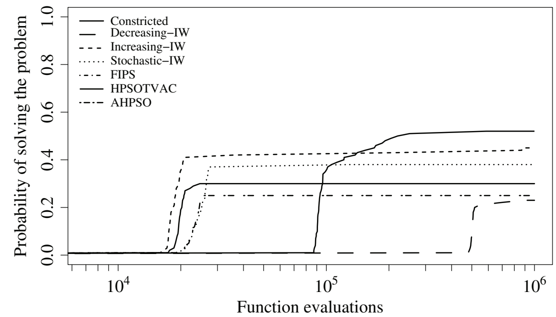
Because of space constraints, we present only one representative example of the results we obtained. Fig. 1 shows some of the algorithms' RLDs when solving Griewank's function. These plots are given with respect to a bound of 0.001% above the optimum value, corresponding to an absolute error of 0.0018. The smallest first hitting times for the same algorithm across different population size and topology settings are obtained with a population size of 20 and the fully connected topology. Conversely, the largest ones are obtained with a population size of 60 and the ring topology. With 20 particles, the right tails of the RLDs show a slowly increasing or a non-increasing slope. This means that, for the Griewank's function, all the PSO variants included in our study, when using 20 particles and the parameter settings shown in Table II, have a strong stagnation tendency. In fact, no variant is capable of finding a solution of the required quality with probability 1.0 with this population size. With 60 particles and a ring topology, only FIPS finds the required solution quality with probability 1.0, while the constricted PSO and HPSOTVAC reach a solution of the required quality with probability 0.99.

Result 1: Depending on the problem and required solution quality, PSO algorithms exhibit a stagnation tendency with different degrees of severity. This tendency is smaller when using large population sizes and/or low connected topologies than it is when using small population sizes and/or highly connected topologies; however, even though the probability of solving the problem increases, first hitting times are normally delayed.

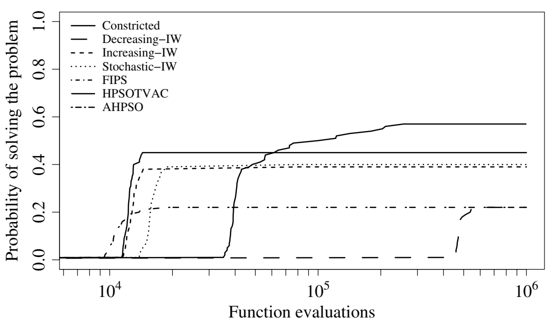
An interesting fact is the strong influence of the topology on the algorithms' performance. For example, FIPS with a fully connected topology does not find a single solution of the required quality; however, with a ring topology, it is among the fastest algorithms (in terms of first hitting time). AHPSO seems to profit from a highly connected topology at the beginning of a run. It is also among the fastest variants when the rest of the algorithms use a von Neumann or ring



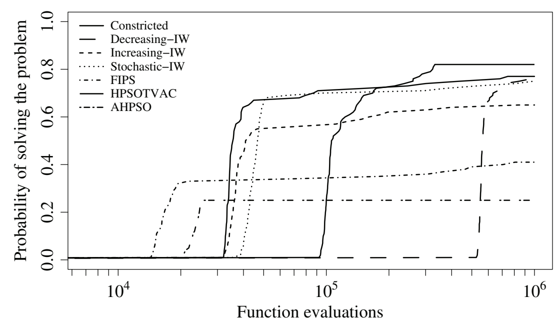
(a) 20 particles, fully connected topology



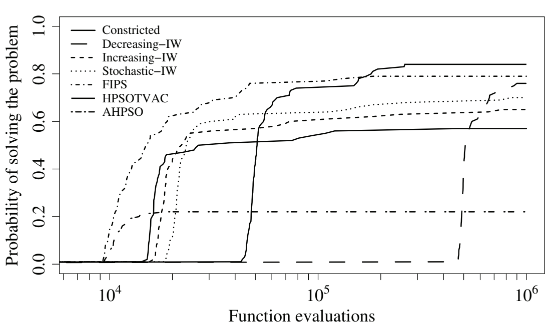
(b) 60 particles, fully connected topology



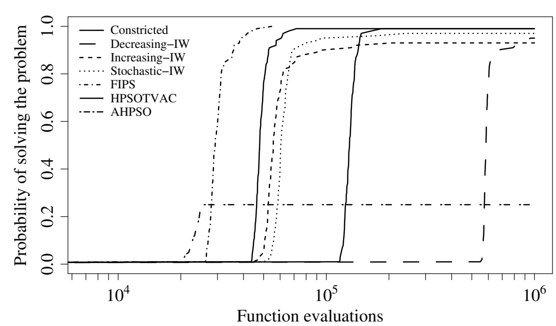
(c) 20 particles, von Neumann topology



(d) 60 particles, von Neumann topology



(e) 20 particles, ring topology



(f) 60 particles, ring topology

Fig. 1. RLDs on Griewank's function. The solution quality bound is set to 0.001% above the global optimum (equivalent to an absolute error of 0.0018). Plots (a), (c), and (e) in the left column show the RLDs obtained with 20 particles. Plots (b), (d), and (f) in the right column show the RLDs obtained with 60 particles. The effect of using different population topologies can be seen by comparing plots in different rows. The effect of using a different number of particles can be seen by comparing columns.

topology. However, it is unable to solve the problem with a high probability.

Result 2: PSO algorithms are sensitive to changes in the population topology in different degrees. Among those tested, FIPS is the most sensitive variant to a change of this nature. On the contrary, HPSOTVAC and the decreasing inertia weight PSO algorithm are quite stable to topology changes.

As a best case analysis, we now consider the possibility of restarting the algorithms with an optimal cut-off period. In Table III, we show the best configuration of each algorithm to solve Griewank's problem (at 0.001% above the global optimum) with probability 0.99. The best performing configurations of FIPS and the constricted PSO, both with 60 particles and the ring topology, do not benefit from restarts under these conditions, and they are the two best variants for the considered goal. In this case, the joint effect of choosing the right algorithm, with an appropriate population size and with the right topology, cannot be outperformed by configurations that benefit the most from restarts (i.e., those that stagnate). Similar analyses were performed on all the problems of our benchmark suite but different results were obtained in each case.

TABLE III

BEST PERFORMING CONFIGURATIONS OF EACH ALGORITHM USING INDEPENDENT RESTARTS ON GRIEWANK'S FUNCTION[1, 2]

Algorithm	Pop. size	Topology	Cut-off	Effort	Restarts
FIPS	60	Ring	46440	**46440**	0
Constricted	60	Ring	71880	71880	0
Sto-IW	40	Ring	52160	131075	2
Inc-IW	20	Ring	24040	138644	5
HPSOTVAC	40	Ring	132080	155482	1
AHPSO	40	Dynamic	17360	207295	11
Dec-IW	60	Ring	663000	1326000	1

[1] Probabilities taken from the RLDs.

[2] Cut-off and effort measured in function evaluations. The effort is computed using (8).

TABLE IV
DISTRIBUTION OF APPEARANCES OF DIFFERENT PSO ALGORITHMS IN THE TOP-THREE GROUP[1]

FES	Ackley	Griewank	Rastrigin	Salomon	Schwefel	Step	Rosenbrock	Sphere
10^3	FIPS (F, vN) Inc-IW (F)	FIPS (F, vN) Inc-IW (F)	FIPS (F, vN) Inc-IW (F)	FIPS (F, vN) HPSOTVAC	Inc-IW (F, vN, R)	FIPS (F, vN) Inc-IW (F)	AHPSO Constricted (F) Sto-IW (F)	FIPS (F, vN) Inc-IW (F)
10^4	FIPS (vN, R) Inc-IW (F)	Constricted (F) FIPS (vN) Inc-IW (F)	AHPSO Constricted (F) Inc-IW (F)	Constricted (F) Inc-IW (F) Sto-IW (F)	AHPSO Inc-IW (F) Sto-IW (F)	AHPSO Constricted (F) Inc-IW (F) Sto-IW (F)	AHPSO Constricted (F) Sto-IW (F)	AHPSO Constricted (F) Inc-IW (F)
10^5	Constricted (vN) FIPS (R) Inc-IW (F)	Constricted (vN, R) FIPS (R) Inc-IW (vN, R) Sto-IW (vN, R)	FIPS (vN) Inc-IW (vN) Sto-IW (vN)	Constricted (vN, R) FIPS (R) Inc-IW (F, vN) Sto-IW (F, vN, R)	HPSOTVAC (F, vN, R)	Constricted (vN) Inc-IW (F) Sto-IW (F)	AHPSO Constricted (F) Sto-IW (F)	AHPSO Constricted (F, vN, R) FIPS (R) Inc-IW (F, vN, R) Sto-IW (F, vN, R)
10^6	Constricted (vN, R) Dec-IW (F, vN, R) FIPS (R) Inc-IW (vN, R) Sto-IW (vN, R)	Constricted (vN, R) Dec-IW (vN, R) FIPS (R) HPSOTVAC (F, vN, R) Inc-IW (vN, R) Sto-IW (vN, R)	HPSOTVAC (F, vN, R)	Constricted (vN, R) Dec-IW (F, vN, R) FIPS (R) HPSOTVAC (F, vN, R) Inc-IW (vN, R) Sto-IW (vN, R)	Dec-IW (vN) FIPS (R) HPSOTVAC (R)	Constricted (vN, R) Dec-IW (F, vN, R) FIPS (R) HPSOTVAC (F, vN, R) Inc-IW (F, vN, R) Sto-IW (F, vN, R)	AHPSO Constricted (F) Sto-IW (F)	AHPSO Constricted (F, vN, R) Dec-IW (F, vN, R) FIPS (R) HPSOTVAC (vN) Inc-IW (F, vN, R) Sto-IW (F, vN, R)

[1] F, vN, and R stand for fully connected, von Neumann, and ring, respectively. FES stands for function evaluations.

Result 3: Independent restarts can improve the performance of various PSO algorithms. In some cases, configurations that favor an exploitative behavior can outperform those that favor an exploratory one if optimal restart policies are used. However, the optimal restart policy is algorithm- and problem-dependent and therefore cannot be defined a priori.

B. Results: Aggregated Data

The analysis that follows is based on the median solution quality achieved by an algorithm after some specific number of function evaluations. This analysis considers only the 40-particle case, which represents the intermediate case in terms of population size in our experimental setup. For each problem, we ranked 19 configurations (6 PSO algorithms × 3 topologies + AHPSO) and selected only those that were ranked in the first three places (what we call the top-three group). For this analysis, we assume that the algorithms are neither restarted nor fine-tuned for any specific problem.

Table IV shows the distribution of appearances of the compared PSO algorithms in the top-three group. The table shows configurations ranked among the three best algorithms for different numbers of function evaluations (FES). The topology used by a particular configuration is shown in parenthesis. If two or more configurations found solutions with the same quality level (differences smaller than 10^{-15} are not considered) and they were among the three best solution qualities, these configurations were considered to be part of the top-three group. In fact, we observed that, as the number of function evaluations increases, more and more algorithms appear in the top-three group. This indicates that the difference in the solution quality achieved by different algorithms decreases and that many algorithms find solutions of the same quality level.

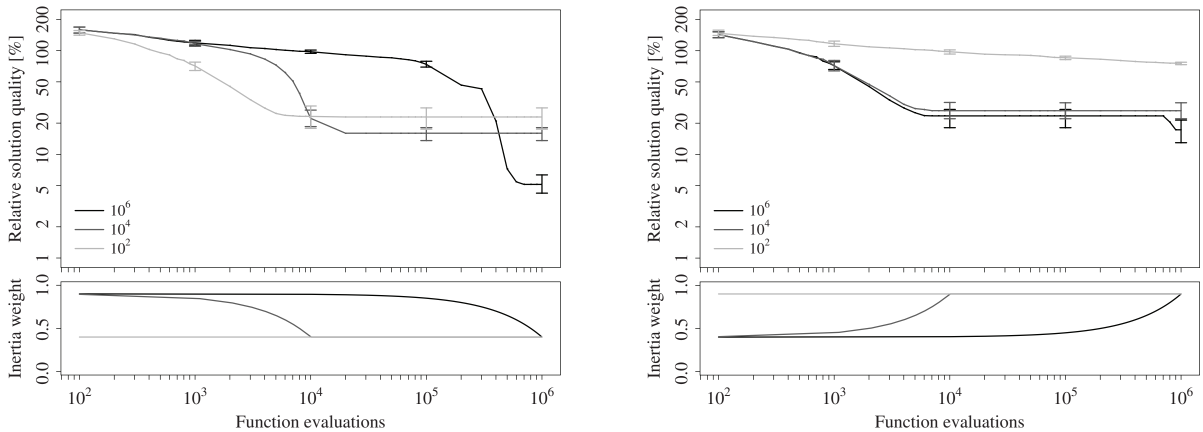
Table V shows the algorithms that most often appear in the top-three group in Table IV for different termination criteria.

TABLE V
BEST PSO VARIANTS FOR DIFFERENT TERMINATION CRITERIA

Budget (in FES)	Algorithm (Topology)	Σ	Multi/unimodal
10^3	Inc-IW(F), FIPS(F, vN)	6	5/1
10^4	Inc-IW(F)	7	6/1
10^5	Constricted(vN)	5	4/1
10^6	Dec-IW(vN), FIPS(R)	6	5/1

The column labeled "Σ" shows the total number of times each algorithm appeared in the top-three group. The rightmost column shows the distribution of appearances in the top-three group between multi- and unimodal functions.

Note that the connectivity of the topology used by the best ranked variants decreases as the maximum number of function evaluations increases. Note also that FIPS is among the best ranked variants: for the shortest runs, using a fully connected or a von Neumann topology and, for the longest runs, using a ring topology. Even though these results may seem counterintuitive at first inspection, they can be understood by looking at the convergence behavior of the algorithm when topologies of different connectivity degree are used. In FIPS, highly connected topologies induce a strongly convergent behavior that, depending on the features of the objective function, can result in a very fast solution improvement during the first iterations [26]. Indeed, it has been shown that under stagnation, the moments of the sampling distribution of FIPS become more and more stable (over time) as the topology connectivity increases [27]. This means that, in FIPS, the more connected the population topology, the lower the stochasticity in the behavior of a particle. By observing the behavior of FIPS over different run lengths, our results extend those of Mendes [21] who studied the behavior of FIPS using only a fixed number of function evaluations as stopping criterion.



(a) Decreasing inertia weight

(b) Increasing inertia weight

Fig. 2. Solution quality and inertia weight development over time for different inertia weight schedules on the Rastrigin function. The solution quality development plots are based on the medians of the algorithms' RLDs. The first and third quartiles are shown at selected points. These results correspond to configurations of 20 particles in a fully connected topology. The results obtained with the schedules of 10^5 and 10^3 function evaluations (not shown) are intermediate with respect to the results obtained with the other schedules.

Result 4: When a limited number of function evaluations are allowed, configurations that favor an exploitative behavior (i.e., those with highly connected topologies and/or low inertia weights) obtain the best results. When solution quality is the most important aspect, algorithms with exploratory properties are the best performing.

C. Results: Different Inertia Weight Schedules

With very few exceptions (e.g., [28]), the change of the inertia weight value in the time-decreasing/increasing inertia weight variants is normally scheduled over the whole optimization process. In this section, we present a study on the effects of using different schedules on both the time-decreasing and time-increasing inertia weight variants. To do so, we modified the inertia weight schedule, which is based on (5), so that whenever the inertia weight reaches its limit value, it remains there. We experimented with five inertia weight schedules of $wt_{\max} \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$ function evaluations each. The remaining parameters were set as shown in Table II.

As an example of the effects of different inertia weight schedules, consider Fig. 2, which shows the development of the solution quality over time (using both the time-decreasing and time-increasing inertia weight variants) for different inertia weight schedules on the Rastrigin function.

In the case of the time-decreasing inertia weight variant, slow schedules ($wt_{\max} = 10^5$ or 10^6 function evaluations) perform poorly during the first phase of the optimization process; however, they are the ones that are capable of finding the best quality solutions. On the other hand, fast schedules ($wt_{\max} = 10^2$ or 10^3 function evaluations) produce rapid improvement but at the cost of stagnation later in the optimization process.

With the time-increasing inertia weight variant, slow schedules provide the best performance. Fast schedules make the time-increasing inertia weight variant strongly stagnant. For both variants, the severity of the stagnation tendency induced by different schedules is alleviated by both an increase in the number of particles and the use of a low connected topology.

Result 5: By varying the inertia weight schedule, it is possible to control the convergence speed of the time-varying inertia weight variants. In the case of the time-decreasing inertia weight variant, faster schedules induce a faster convergence speed, albeit at the cost of increasing the algorithm's stagnation tendencies. In the time-increasing inertia weight variant, slow schedules provide the best performance both in terms of speed and quality.

D. Summary

The goal of the comparison presented in this section was to identify algorithmic components that provide good performance under different operating conditions (especially run lengths). The five main results give insight into which factors should be taken into account when trying to solve effectively a problem using a PSO algorithm.

Among other results, we have seen that the stagnation tendency of PSO algorithms can be alleviated by using a large population and/or a low connected topology. Another approach to reduce stagnation in some cases is to use restarts. However, optimal restart schedules are algorithm and problem dependent and determining them requires previous experimentation. We have also seen how different inertia weight schedules affect the performance of the time-decreasing/increasing inertia weight variants.

V. Frankenstein's Particle Swarm Optimization Algorithm

Insights on experimental results ideally guide toward the definition of new better performing algorithms. In this section, a composite algorithm called Frankenstein's PSO is assembled from algorithmic components that are taken from the PSO algorithms that we have examined or that are derived from the analysis of the comparison results.

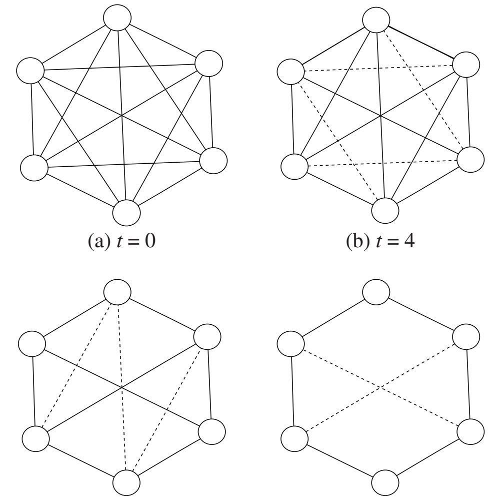
Algorithm 1 Frankenstein's PSO algorithm

/* Initialization */
for $i = 1$ to n **do**
  Create particle p_i and add it to the set of particles $\mathcal{P}$
  Initialize its vectors $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$ to random values within the search range and maximum allowed velocities
  Set $\boldsymbol{pb}_i = \boldsymbol{x}_i$
  Set $\mathcal{N}_i = \mathcal{P}$
end for

/* Main Loop */
Set $t = 0$
Set $esteps = 0$
repeat
  /* Evaluation Loop */
  for $i = 1$ to n **do**
    if $f(\boldsymbol{x}_i)$ is better than $f(\boldsymbol{pb}_i)$ **then**
      Set $\boldsymbol{pb}_i = \boldsymbol{x}_i$
    end if
  end for
  /* Topology Update */
  if $t > 0 \wedge t <= k \wedge t \bmod \lceil k/(n-3) \rceil = 0$ **then**
    /* $t > 0$ ensures that a fully connected topology is used first */
    /* $t <= k$ ensures that the topology update process is not called after iteration k */
    /* $t \bmod \lceil k/(n-3) \rceil = 0$ ensures the correct scheduling of the topology update process */
    for $i = 1$ to $n - (2 + esteps)$ **do**
      /* $n - (2 + esteps)$ ensures the arithmetic regression pattern */
      if $|\mathcal{N}_i| > 2$ **then**
        /* $|\mathcal{N}_i| > 2$ ensures proper node selection */
        Select at random particle p_r from $\mathcal{N}_i$ such that p_r is not adjacent to p_i
        Eliminate particle p_r from $\mathcal{N}_i$
        Eliminate particle p_i from $\mathcal{N}_r$
      end if
    end for
    Set $esteps = esteps + 1$
  end if
  /* Inertia Weight Update */
  if $t \leq wt_{\max}$ **then**
    Set $w^t = \frac{wt_{\max} - t}{wt_{\max}}(w_{\max} - w_{\min}) + w_{\min}$
  else
    Set $w^t = w_{\min}$
  end if
  /* Velocity and Position Update */
  for $i = 1$ to n **do**
    Generate $\boldsymbol{U}_m^t \; \forall p_m \in \mathcal{N}_i$
    Set $\varphi_m = \varphi/|\mathcal{N}_i| \; \forall p_m \in \mathcal{N}_i$
    Set $\boldsymbol{v}_i^{t+1} = w^t \boldsymbol{v}_i^t + \sum_{p_m \in \mathcal{N}_i} \varphi_k \boldsymbol{U}_k^t (\boldsymbol{pb}_k^t - \boldsymbol{x}_i^t)$
    Set $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}$
  end for
  Set $t = t + 1$
  Set $\boldsymbol{sol} = \underset{p_i \in \mathcal{P}}{\operatorname{argmin}} f(\boldsymbol{pb}_i^t)$
until $f(\boldsymbol{sol})$ value is good enough or $t = t_{\max}$

A. Algorithm

Frankenstein's PSO is composed of three main algorithmic components, namely, 1) a time-varying population topology that reduces its connectivity over time, 2) the FIPS mechanism for updating a particle's velocity, and 3) a decreasing inertia weight. These components are taken from AHPSO, FIPS, and the time-decreasing inertia weight variant, respectively. The first component is included as a mechanism for improving the tradeoff between speed and quality associated with topologies of different connectivity degrees. The second component is used because the analysis showed that FIPS is the only algorithm that can outperform the others using topologies of different connectivity degree (see Table V). Finally, the decreasing inertia weight component is included as a mean to balance the exploration-exploitation behavior of the algorithm.

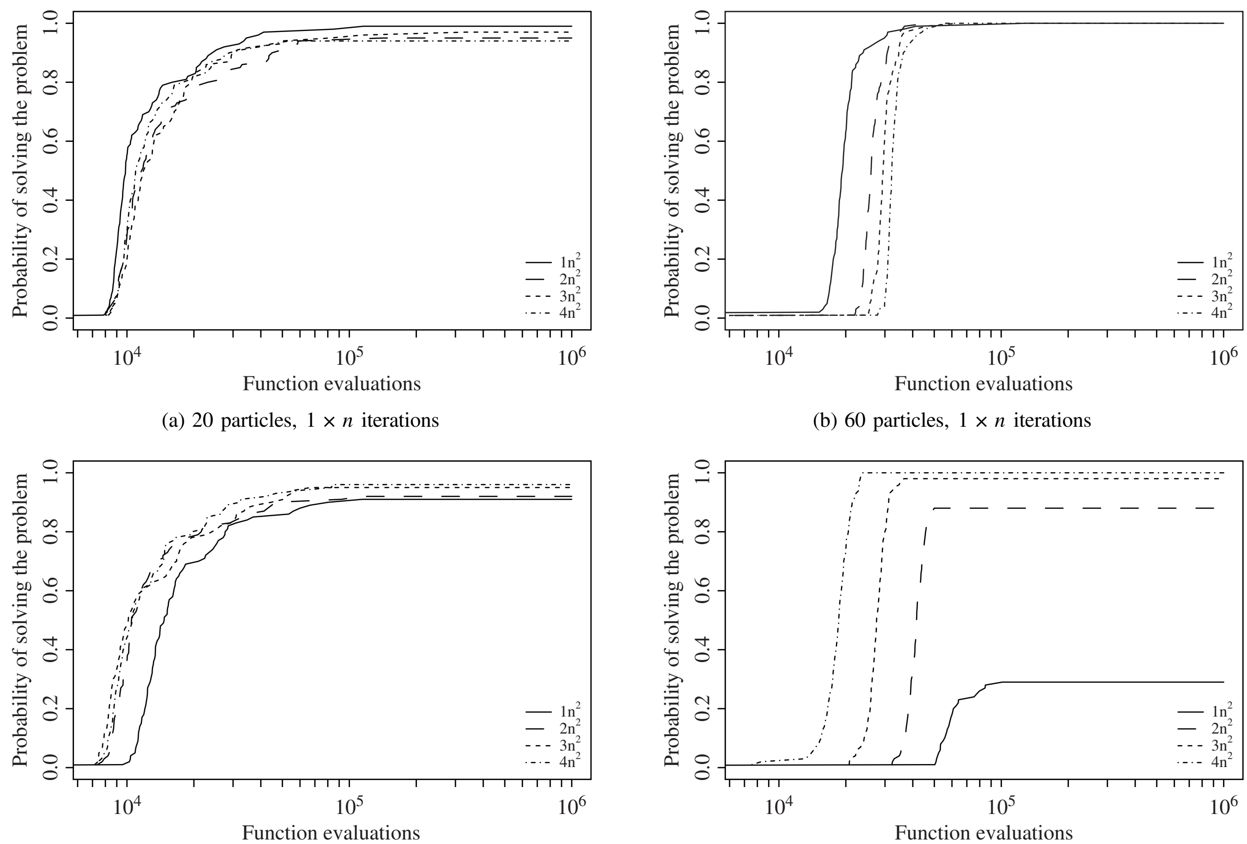


(a) $t = 0$ (b) $t = 4$ (c) $t = 8$ (d) $t = 12$

Fig. 3. Topology change process. Suppose $n = 6$ and $k = 12$. Then, every $\lceil 12/(6-3) \rceil = 4$ iterations we remove some edges from the graph. In $6-3 = 3$ steps, the elimination process will be finished. (a) At $t = 0$ a fully connected topology is used, (b) At $t = 4$ the $6 - 2 = 4$ edges to be removed are shown in dashed lines, (c) At $t = 8$ the $6 - 3 = 3$ edges to be removed are shown in dashed lines, and (d) At $t = 12$ the remaining $6 - 4 = 2$ edges to be removed are shown in dashed lines. From $t = 12$ on, the algorithm uses a ring topology.

The time-varying topology starts as a fully connected one and, as the optimization process evolves, decreases its connectivity until it ends up being a ring topology. Interestingly, it is the opposite approach to the one taken by Suganthan [29]. Note, however, that our approach is entirely based on the results of the empirical analysis presented in the previous section. Specifically, our choice is based on the fact that a highly connected topology during the first iterations gives an algorithm the opportunity to find good quality solutions early in a run (see Table V and Results 1 and 4 in Section IV). The topology connectivity is then decreased, so that the risk of getting trapped somewhere in the search space is reduced and, hence, exploration is enhanced. Including this component into the algorithm allows it to achieve good performance across a wider range of run lengths as it will be shown later. As we said before, this component is taken from AHPSO. Information flow in AHPSO is very fast during the first iterations because the topology connectivity is high. As the optimization process evolves, its connectivity decreases.

In Frankenstein's PSO, we do not use a hierarchical topology, as it is not clear from our results how it contributes to a good performance. Instead, the topology is changed as follows. Suppose we have a particle swarm composed of n particles. We schedule the change of the topology so that in k iterations (with $k \geq n$), we transform a fully connected topology with $n(n-1)/2$ edges into a ring topology with n edges. The total number of edges that have to be eliminated is $n(n-3)/2$. Every $\lceil k/(n-3) \rceil$ iterations we remove m edges, where m follows an arithmetic regression pattern of the form $n-2, n-3, \ldots, 2$. We sweep m nodes removing one edge per node. The edge



(a) 20 particles, $1 \times n$ iterations

(b) 60 particles, $1 \times n$ iterations

(c) 20 particles, $4 \times n$ iterations

(d) 60 particles, $4 \times n$ iterations

Fig. 4. RLDs obtained by Frankenstein's PSO algorithm on Griewank's function. The solution quality demanded is 0.001% above the global optimum. Each graph shows four RLDs that correspond to different inertia weight schedules.

to be removed is chosen uniformly at random from the edges that do not belong to the exterior ring, which is predefined (just as it is done when using the normal ring topology). The transformation from the initially fully connected to the final ring topology is performed in $n - 3$ elimination steps. Fig. 3 shows a graphical example of how the process just described is carried out.

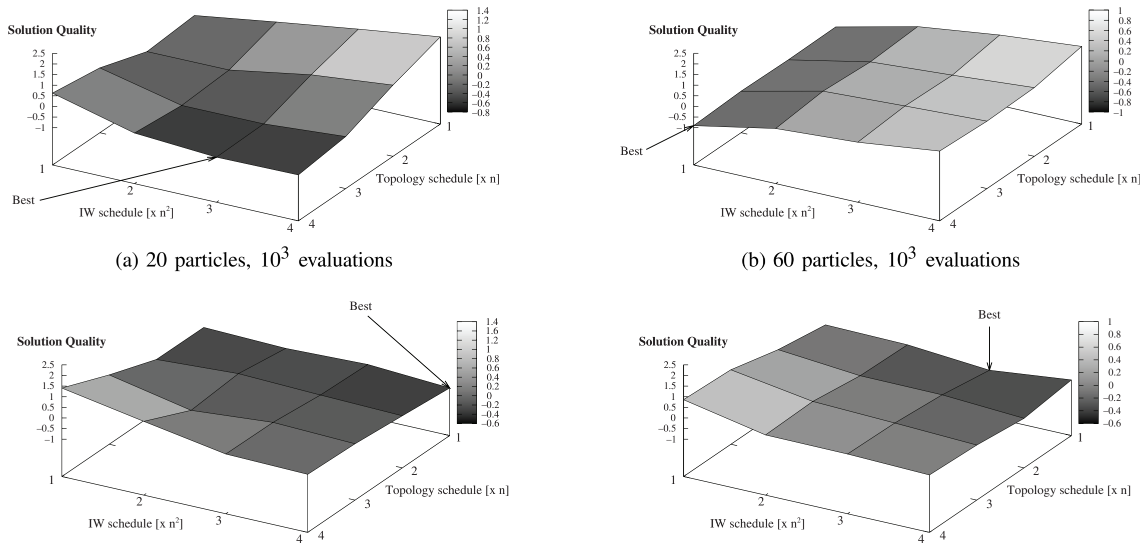
Changes in the population topology must be exploited by the underlying particles' velocity-update mechanism. In Frankenstein's PSO we included the mechanism used by FIPS. The reason for this is that we need a component that offers good performance across different topology connectivities. According to Table V, the only velocity-update mechanism that is ranked among the best variants when using different topologies is the one used by FIPS. For short runs, the best performance of FIPS is obtained with the fully connected topology (the way Frankenstein's PSO topology starts); for long runs, FIPS reaches very high performance with a low connected topology (the way Frankenstein's PSO topology ends).

The constriction factor originally used in FIPS is substituted by a decreasing inertia weight. A decreasing inertia weight was chosen because it is a parameter that can be used to control the algorithm's exploration/exploitation capabilities. In Section IV-C, we saw that a proper selection of the inertia weight schedule can dramatically change the performance of a PSO algorithm. A decreasing inertia weight would counterbalance the exploratory behavior that the chosen topology change scheme could induce.

The pseudocode of Frankenstein's PSO is shown in Algorithm 1. The main loop cycles through the three algorithmic components: topology update, inertia weight update, and the particles' velocity and position updates. The topology update mechanism is only executed while the algorithm's current number of iterations is lower than or equal to a parameter k, which specifies the topology update schedule. Since it is guaranteed that the ring topology is reached after iteration k, there is no need to call this procedure thereafter. In Algorithm 1, a variable *esteps* is used to ensure that the number of eliminated edges in the topology follows an arithmetic regression pattern. Note that the elimination of neighborhood relations is symmetrical; that is, if particle r is removed from the neighborhood of particle i, particle i is also removed from the neighborhood of particle r. The inertia weight is then updated, and finally, the velocity-update mechanism is applied in the same way as in FIPS.

B. Parameterization Effects

We studied the impact of using different schedules for the topology and inertia weight updates on the algorithm's performance. The remaining parameters were set as follows: the maximum velocity $V_{\max}$ is set to $\pm X_{\max}$ (the maximum of the search range), the linearly-decreasing inertia weight



(a) 20 particles, 10^3 evaluations

(b) 60 particles, 10^3 evaluations

(c) 20 particles, 10^6 evaluations

(d) 60 particles, 10^6 evaluations

Fig. 5. Average standard solution quality as a function of the topology update and the inertia weight schedules for different termination criteria. In each case, the best configuration is pointed out by an arrow.

is varied from 0.9 to 0.4, and the sum of the acceleration coefficients φ is set to 4.0.

The experimental conditions described in Section III are used. Three swarm sizes ($n = 20,\ 40,\ 60$), four schedules of the topology update (measured in iterations; $k = n,\ 2n,\ 3n,\ 4n$) and four schedules of the inertia weight (measured in function evaluations; $wt_{\max} = n^2,\ 2n^2,\ 3n^2,\ 4n^2$) were tried. Note that the values of k and $wt_{\max}$ are independent of each other.

As an illustrative example of the results, consider Fig. 4. It shows the RLDs obtained by Frankenstein's PSO algorithm on Griewank's function. These distributions correspond, as before, to a solution quality 0.001% above the optimum value. Only the results obtained with 4 out of the 12 possible combinations of topology schedules and population sizes are shown.[2]

A combination of a slow topology update schedule ($3n$ or $4n$) and a fast inertia weight schedule (n^2 or $2n^2$) promotes the stagnation of the algorithm. This can be explained if we recall that FIPS has a strong stagnation tendency when using a highly connected topology: A slow topology update schedule maintains a high topology connectivity for more iterations, and a fast inertia weight schedule quickly reduces the exploration capabilities of the particle swarm. These two effects also increase the algorithm's stagnation tendency. To counteract a fast stagnation tendency, the two possibilities are to slow down the inertia weight schedule or to speed up the change of the topology.

Increasing the number of particles increases the amount of information available to the algorithm during the first iterations. The exploitation of this information depends on the topology update and inertia weight schedules. The configurations that appear to better exploit it are those in which these two schedules are slow.

To compare the configurations' relative performance across problems that have different scales, we look at the average (over the eight benchmark problems of the experimental setup) of the standardized median solution quality (i.e., for each group, the mean is equal to zero and the standard deviation is equal to 1) as a function of the topology update and the inertia weight schedules for different termination criteria. The results are shown in Fig. 5. Since we work with minimization problems, a lower average standard solution quality means that the specific configuration found better solutions.

According to Fig. 5, the algorithm needs more exploratory configurations (i.e., fast topology update schedules and slow inertia weight schedules) for long runs. For short runs, configurations with slow topology update schedules and fast inertia weight schedules yield the best results. For runs of 10^4 and 10^5 function evaluations, the best configurations are intermediate ones (i.e., fast or slow schedules for both the topology and inertia weight updates).

The more exploratory behavior that a large population provides needs to be counterbalanced by the chosen configuration. For example, at 10^3 function evaluations, the best configuration tends to have faster inertia weight schedules for larger swarms. With 20 particles, the best configuration is at point $(4, 3)$, while with 40 and 60 particles the best configurations are at $(4, 2)$ and $(4, 1)$, respectively. These results are consistent with those of the experimental comparison.

Like any other algorithm, Frankenstein's PSO has its own set of parameters that need to be set by the practitioner before trying to solve a problem. The final parameter settings will depend on the class of problems one is trying to solve and on the application scenario requirements. Based on the results presented in this section we can derive the following guidelines for choosing the topology and the inertia weight schedules. If the number of function evaluations is restricted, a configuration with 20 particles, a slow topology change schedule ($\approx 4n$), and an intermediate inertia weight schedule

[2] We remind the reader that the full experimental data are available at [20].

TABLE VI

BEST OVERALL CONFIGURATIONS OF DIFFERENT PSO VARIANTS FOR DIFFERENT TERMINATION CRITERIA. EACH GROUP IS SORTED BY THE AVERAGE STANDARD SOLUTION QUALITY IN ASCENDING ORDER, SO THE BEST OVERALL CONFIGURATION IS LISTED FIRST

FES	Algorithm	Ackley	Griewank	Rastrigin	Salomon	Schwefel	Step	Rosenbrock	Sphere	Average
10^3	Frankenstein's PSO	**−2.024**	**−0.955**	−0.975	**−0.517**	1.378	**−1.315**	−0.302	**−1.108**	−0.727
	Increasing-IW	−0.013	−0.393	−0.950	−0.323	**−1.229**	−0.645	−0.367	−0.371	−0.536
	Decreasing-IW	−0.002	−0.386	**−1.067**	−0.316	−1.199	−0.359	−0.474	−0.425	−0.528
	FIPS	−0.765	−0.430	−0.080	−0.457	1.432	−0.932	0.206	−0.538	−0.195
	Constricted	0.476	−0.156	0.287	−0.276	−0.213	0.406	**−0.491**	−0.057	−0.003
	Stochastic-IW	0.656	0.124	0.652	−0.237	−0.046	0.693	−0.488	0.304	0.207
	AHPSO	0.476	−0.156	0.287	2.464	−0.213	0.406	**−0.491**	−0.057	0.340
	HPSOTVAC	1.198	2.353	1.847	−0.338	0.090	1.745	2.406	2.251	1.444
10^4	Increasing−IW	−0.129	−0.564	−0.593	−0.349	**−0.797**	−0.539	−0.348	−0.359	−0.460
	Constricted	−0.212	−0.616	−0.591	−0.373	−0.459	−0.539	−0.376	−0.359	−0.441
	Decreasing-IW	−0.065	−0.518	**−0.962**	−0.341	−0.754	−0.085	−0.370	−0.358	−0.431
	Frankenstein's PSO	**−1.061**	**−0.761**	0.056	**−0.386**	1.332	**−0.993**	**−0.414**	**−0.361**	−0.324
	Stochastic-IW	−0.131	0.443	−0.512	−0.361	−0.541	−0.085	−0.290	−0.359	−0.230
	FIPS	−1.056	−0.718	1.567	−0.378	1.760	−0.539	−0.364	**−0.361**	−0.011
	AHPSO	0.569	0.656	−0.512	2.474	−0.641	0.596	−0.312	−0.316	0.314
	HPSOTVAC	2.086	2.077	1.546	−0.287	0.101	2.185	2.473	2.475	1.582
10^5	Frankenstein's PSO	**−0.354**	**−0.883**	−1.192	**−0.359**	**−1.548**	−0.487	0.782	**−0.354**	−0.549
	Decreasing-IW	**−0.354**	0.631	−0.709	−0.355	−0.311	**−0.787**	−0.983	**−0.354**	−0.402
	Increasing-IW	**−0.354**	0.631	0.108	−0.355	−0.271	**−0.787**	−0.441	**−0.354**	−0.228
	Constricted	**−0.354**	**−0.883**	0.313	**−0.359**	0.729	−0.487	0.216	**−0.354**	−0.147
	Stochastic-IW	**−0.354**	0.631	1.130	**−0.359**	0.649	**−0.787**	−1.013	**−0.354**	−0.057
	FIPS	**−0.354**	**−0.883**	1.060	−0.355	1.372	0.712	1.008	**−0.354**	0.276
	AHPSO	**−0.354**	1.639	0.721	2.475	0.529	0.712	**−1.019**	**−0.354**	0.544
	HPSOTVAC	2.475	**−0.883**	**−1.431**	−0.334	−1.149	1.911	1.449	2.475	0.564
10^6	Frankenstein's PSO	**−0.354**	**−0.354**	−0.787	**−0.358**	−1.257	**−0.661**	−0.058	**−0.504**	−0.542
	Increasing-IW	**−0.354**	**−0.354**	0.002	−0.354	0.019	**−0.661**	0.039	**−0.504**	−0.271
	Decreasing-IW	**−0.354**	**−0.354**	0.472	−0.354	0.367	**−0.661**	**−0.778**	**−0.504**	−0.271
	FIPS	**−0.354**	**−0.354**	−0.546	−0.354	**−1.349**	0.661	0.685	**−0.504**	−0.264
	Stochastic-IW	**−0.354**	**−0.354**	0.415	**−0.358**	0.705	**−0.661**	−0.529	**−0.504**	−0.205
	Constricted	**−0.354**	**−0.354**	0.815	**−0.358**	1.072	**−0.661**	−0.717	**−0.504**	−0.132
	HPSOTVAC	2.475	**−0.354**	**−1.760**	−0.341	−0.705	0.661	2.129	2.184	0.536
	AHPSO	**−0.354**	2.475	1.388	2.475	1.149	1.984	−0.771	0.840	1.148

($\approx 3n^2$) would be the first one to try. If solution quality is the main concern, a configuration with 60 particles, a fast topology update schedule ($\approx n$), and a slow inertia weight ($\approx 4n^2$) should be preferred.

VI. PERFORMANCE EVALUATION

The performance of Frankenstein's PSO is evaluated by comparing its best configurations with those of the PSO algorithms described in Section IV. The best configurations of each variant were selected using the same ranking scheme as in Section IV-B. The list of selected configurations is available at [20].

Table VI shows the standardized median solution quality obtained by each configuration (identified only by the algorithm's name) for each termination criterion. The best values for each individual problem and stopping criterion are highlighted in boldface.

For runs of 10^3, 10^5, and 10^6 function evaluations, the best overall configuration is the one of Frankenstein's PSO. For runs of 10^4 function evaluations, the configuration of Frankenstein's PSO is ranked in the fourth place. However, with this same number of function evaluations, the configuration of Frankenstein's PSO is the best configuration in six of the eight benchmark problems. The average rank of Frankenstein's PSO after 10^4 function evaluations can be explained with the results on Schwefel's function: FIPS (of which a component is used in Frankenstein's PSO) is the worst algorithm for this termination criterion (and also for the one of 10^3 function evaluations) on Schwefel's function.

The performance of Frankenstein's PSO suggests that indeed it is possible and profitable to integrate different existing algorithmic components into a single PSO variant. The results show that by composing existing algorithmic components, new high-performance variants can be built. At the same time, it is possible to gain insights into the effects of the interactions of different components on the algorithm's final performance. Of course, just as it is possible to take advantage of the strengths of different components, it is also possible that their weaknesses are passed on: the performance of Frankenstein's PSO on Schwefel's function is an example of this.

VII. CONCLUSION AND FUTURE WORK

Many PSO variants are proposed in the current literature. This is a consequence of the great attention that PSO has received since its introduction. However, it is also a sign of the lack of knowledge about which algorithmic components provide good performance on particular types of problems and under different operating conditions.

In an attempt to gain insight into the performance advantages that different algorithmic components provide, we compared what we consider to be some of the most influential or promising PSO variants. For practical reasons, many variants were left out of this paper. Future studies should consider other variants as well as other components that are not necessarily present in existing PSO algorithms. In fact, some works are already exploring these issues [30]–[34]. Recently, an alternative way of composing algorithmic components has been proposed [35]. The approach consists in shifting the integration of components from the particle level to the swarm level by creating heterogeneous swarms, that is, swarms composed of particles that move using different rules (i.e., algorithmic components). An avenue of research that seems promising is to experiment with random topologies that satisfy some constraints (e.g., a desired average connection degree). These works would help in improving our understanding of the interactions among PSO algorithmic components.

As might be expected, the results of our experimental comparison showed that no variant dominates all the others on all the problems of our benchmark suite over different run lengths. Nevertheless, we were able to identify general trends on the influence that various PSO algorithmic components and their parameters have on performance.

Based on these insights, we explored the possible advantages of combining algorithmic components that provided good performance into a single PSO variant by assembling a composite algorithm that we call Frankenstein's PSO. This new PSO algorithm is composed of three main algorithmic components: 1) a time-varying population topology that decreases its connectivity as the optimization process evolves; 2) a particles' velocity-update mechanism that exploits every stage of the topology change process; and 3) a time-decreasing inertia weight that allows the user to tune the algorithm's exploration/exploitation capabilities. In many cases, Frankenstein's PSO is capable of performing better than the variants from which its components were taken.

As a methodological approach, in-depth experimental studies can help in identifying positive and negative (in terms of performance) interactions among algorithmic components and provide strong guidance for the informed design of new composite algorithms. Another selection of PSO variants would have probably ended up in a different Frankenstein's PSO algorithm. For this reason, further research is needed to understand which components are better suited for particular classes of problems and operating conditions and whether some components can be integrated into the same composite algorithm or not. Methods to quantify the contribution of each component on the composite algorithms' final performance are also needed to achieve this goal.

REFERENCES

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[2] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. 1st ed. Chichester, U.K.: Wiley, 2005.

[3] M. Clerc, *Particle Swarm Optimization*. 1st ed. London, U.K.: Wiley-ISTE, 2006.

[4] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization. An overview," *Swarm Intell.*, vol. 1, no. 1, pp. 33–57, 2007.

[5] M. Clerc and J. Kennedy, "The particle swarm–explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, Feb. 2002.

[6] S.-K. S. Fan and E. Zahara, "A hybrid simplex search and particle swarm optimization for unconstrained optimization," *Eur. J. Oper. Res.*, vol. 181, no. 2, pp. 527–548, 2007.

[7] Y. Shi and R. Eberhart, "Parameter selection in particle swarm optimization," in *Proc. 7th Int. Conf. Evol. Program.*, LNCS vol. 1447. 1998, pp. 591–600.

[8] R. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proc. IEEE Congr. Evol. Comput. 2000*, pp. 84–88.

[9] I. C. Trelea, "The particle swarm optimization algorithm: Convergence analysis and parameter selection," *Inform. Process. Lett.*, vol. 85, no. 6, pp. 317–325, 2003.

[10] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1998, pp. 69–73.

[11] Y. Shi and R. Eberhart, "Empirical study of particle swarm optimization," in *Proc. IEEE Congr. Evol. Comput. 1999*, pp. 1945–1950.

[12] Y.-L. Zheng, L.-H. Ma, L.-Y. Zhang, and J.-X. Qian, "On the convergence analysis and parameter selection in particle swarm optimization," in *Proc. IEEE Int. Conf. Mach. Learning Cybern. 2003*, pp. 1802–1807.

[13] Y.-L. Zheng, L.-H. Ma, L.-Y. Zhang, and J.-X. Qian, "Empirical study of particle swarm optimizer with an increasing inertia weight," in *Proc. IEEE Congr. Evol. Comput. 2003*, pp. 221–226.

[14] R. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *Proc. IEEE Congr. Evol. Comput. 2001*, pp. 94–100.

[15] R. Mendes, J. Kennedy, and J. Neves, "The fully informed particle swarm: Simpler, maybe better," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 204–210, Jun. 2004.

[16] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 240–255, Jun. 2004.

[17] S. Janson and M. Middendorf, "A hierarchical particle swarm optimizer and its adaptive variant," *IEEE Trans. Syst., Man, Cybern. B*, Cybern., vol. 35, no. 6, pp. 1272–1282, Dec. 2005.

[18] J. J. Liang, P. N. Suganthan, and K. Deb, "Novel composition test functions for numerical global optimization," in *Proc. IEEE Swarm Intell. Symp.*, 2005, pp. 68–75.

[19] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Nanyang Technological Univ., Singapore and IIT, Kanpur, India, Tech. Rep. 2005005, 2005.

[20] M. A. Montes de Oca, T. Stützle, M. Birattari, and M. Dorigo. (2008, July). *Frankenstein's PSO: Complete Data* [Online]. Available: http://iridia.ulb.ac.be/supp/IridiaSupp2007-002/

[21] R. Mendes, "Population topologies and their influence in particle swarm performance," Ph.D. dissertation, Escola de Engenharia, Universidade do Minho, Portugal, 2004.

[22] A. Mohais, R. Mendes, C. Ward, and C. Posthoff, "Neighborhood restructuring in particle swarm optimization," in *Proc. 18th Australian Joint Conf. Artificial Intell.*, LNCS vol. 3809. 2005, pp. 776–785.

[23] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. 1st ed. San Francisco, CA: Morgan Kaufmann, 2004.

[24] J. Niehaus and W. Banzhaf, "More on computational effort statistics for genetic programming," in *Proc. 6th Eur. Conf. Genetic Program. EuroGP 2003*, LNCS vol. 2610. pp. 164–172.

[25] M. A. Montes de Oca, T. Stützle, M. Birattari, and M. Dorigo, "A comparison of particle swarm optimization algorithms based on runlength distributions," in *Proc. 5th Int. Workshop, Ant Colony Optimization Swarm Intell. (ANTS '06)*, LNCS vol. 4150, 2006, pp. 1–12.

[26] M. A. Montes de Oca and T. Stützle, "Convergence behavior of the fully informed particle swarm optimization algorithm," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2008, pp. 71–78.

[27] R. Poli, "On the moments of the sampling distribution of particle swarm optimisers," in *Proc. Workshop Particle Swarm Optimization: 2nd Decade. Genetic Evol. Comput. Conf. (GECCO)*. 2007, pp. 2907–2914.

[28] J. Wang and D. Wang, "Experiments and analysis on inertia weight in particle swarm optimization," in *Proc. Int. Conf. Service Syst. Manage.*, 2004, pp. 655–659.

[29] P. N. Suganthan, "Particle swarm optimiser with neighbourhood operator," in *Proc. IEEE Congr. Evol. Comput. 1999*, pp. 1958–1962.

[30] R. Mendes and J. Kennedy, "Stochastic barycenters and beta distribution for gaussian particle swarms," in *Proc. Portuguese Conf. Artificial Intell. (EPIA 2007)*, LNAI vol. 4874, pp. 259–270.

[31] J. Jordan, S. Helwig, and R. Wanka, "Social interaction in particle swarm optimization, the ranked FIPS and adaptive multi-swarms," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2008, pp. 49–56.

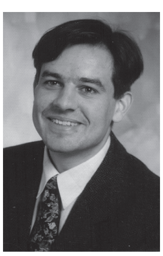
[32] J. Yisu, J. Knowles, L. Hongmei, L. Yizeng, and D. B. Kell, "The landscape adaptive particle swarm optimizer," *Appl. Soft Comput.*, vol. 8, no. 1, pp. 295–304, 2008.

[33] G. Ramana Murthy, M. Senthil Arumugam, and C. K. Loo, "Hybrid particle swarm optimization algorithm with fine tuning operators," *Int. J. Bio-Inspired Comput.*, vol. 1, no. 1–2, pp. 14–31, 2009.

[34] A. Garca-Villoria and R. Pastor, "Introducing dynamic diversity into a discrete particle swarm optimization," *Comput. Oper. Res.*, vol. 36, no. 3, pp. 951–966, 2009.

[35] M. A. Montes de Oca, J. Peña, T. Stützle, C. Pinciroli, and M. Dorigo, "Heterogeneous particle swarm optimizers," in *Proc. IEEE Congr. Evol. Comput. (CEC '09)*. Piscataway, NJ: IEEE Press, pp. 698–705.

Marco A. Montes de Oca received the B.S. degree in computer systems engineering from Escuela Superior de Cómputo, Instituto Politécnico Nacional, Mexico City, Mexico, in 2001, and the M.S. degree in intelligent systems with honors from the Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey, Monterrey, Mexico, in 2005. He is currently working toward the Ph.D. degree in engineering sciences in the Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Université Libre de Bruxelles, Brussels, Belgium.



Thomas Stützle received the Diploma, M.S. degree, in business engineering from the Universität Karlsruhe (TH), Karlsruhe, Germany, in 1994, and the Ph.D. degree and the "Habilitation" in computer science from the Department of Computer Science, the Technische Universität Darmstadt, Germany in 1998 and 2004, respectively.

He is currently a Research Associate of the Belgian F.R.S.-FNRS working in the Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Université Libre de Bruxelles, Brussels, Belgium. He is the author of two books: *Stochastic Local Search: Foundations and Applications* (Morgan Kaufmann), and *Ant Colony Optimization* (MIT Press). He has published extensively in the wider area of metaheuristics (about 100 peer-reviewed articles in journals, conferences, or edited books). His research interests range from stochastic local search (SLS) algorithms, large-scale experimental studies, and automated design of algorithms, to SLS algorithms engineering. He is on the editorial boards of five journals and he has co-edited five journal special issues and eleven workshop or conference proceedings. He has been the main organizer of the workshops on Engineering Stochastic Local Search Algorithms, Brussels, Belgium, in September 2007 and 2009.

Mauro Birattari (M'05) received the Master's degree in electrical and electronic engineering from Politecnico di Milano, Milan, Italy, in 1997, and the Ph.D. degree in information technologies from the Faculty of Engineering, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Université Libre de Bruxelles, Brussels, Belgium, in 2004.

He is currently with IRIDIA-CoDE, Université Libre de Bruxelles, as a Research Associate sponsored by the fund for scientific research F.R.S.-FNRS of Belgium's French community. He has coauthored about 70 peer-reviewed scientific publications in the field of computational intelligence. His research interests focus on swarm intelligence, ant colony optimization, and the automatic design of algorithms. He is an Associate Editor of *Swarm Intelligence* and has served on the organizing committees of a number of international conferences on swarm intelligence, evolutionary algorithms, and metaheuristics, including ANTS, SLS, and GECCO.

Marco Dorigo (S'92–M'93–SM'96–F'06) received the Laurea, Master of Technology, degree in industrial technologies engineering in 1986, and the Ph.D. degree in electronic engineering in 1992 from the Politecnico di Milano, Milan, Italy, and the title of Agrégé de l'Enseignement Supérieur, from the Université Libre de Bruxelles, Brussels, Belgium, in 1995.

From 1992 to 1993, he was a Research Fellow at the International Computer Science Institute, Berkeley, CA. In 1993, he was a NATO-CNR Fellow, and from 1994 to 1996, a Marie Curie Fellow. Since 1996, he has been a tenured Researcher of the FNRS, the Belgian National Funds for Scientific Research, and a Research Director of IRIDIA, the artificial intelligence laboratory of the Université Libre de Bruxelles. He is the inventor of the ant colony optimization metaheuristic. His current research interests include swarm intelligence, swarm robotics, and metaheuristics for discrete optimization. He is the Editor-in-Chief of *Swarm Intelligence*, and an Associate Editor or member of the Editorial Boards of many journals on computational intelligence and adaptive systems.

Dr. Dorigo is a Fellow of the ECCAI. He was awarded the Italian Prize for Artificial Intelligence in 1996, the Marie Curie Excellence Award in 2003, the Dr. A. De Leeuw-Damry-Bourlart award in applied sciences in 2005, and the Cajastur International Prize for Soft Computing in 2007.